Naval Research Laboratory
Washington, DC 20375-5320

NRL/MR/5707--00-8477

Cross-Platform Development: A Difficult Necessity

A Research Project into Cross-Platform Development Tools and Techniques

MICHAEL PILONE
GREGORY STERN
BRIAN SOLAN

Effectiveness of Navy Electronic Warfare Systems (ENEWS/COMSIM) Program
Tactical Electronic Warfare Division

September 29, 2000

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 29, 2000	Final

4. TITLE AND SUBTITLE

Cross-Platform Development: A Difficult Necessity
A Research Project into Cross-Platform Development Tools and Techniques

5. FUNDING NUMBERS

PE — 0602270N
PR — EW70103

6. AUTHOR(S)

Michael Pilone, Gregory Stern, and Brian Solan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/5707--00-8477

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This document provides a general overview of cross-platform development using the C++ programming language, including motivating factors, the problems involved, and simple solutions to the most common pitfalls. This document will also compare and contrast some of the most common commercial and freeware development toolkits which are advertised as cross-platform solutions and discuss how to fill in the elements these toolkits don't provide. This document approaches cross-platform development using the single source cross-platform development model.

14. SUBJECT TERMS

Cross-platform
C++
SGI / NT / Linux
Toolkit
Compare
Evaluation

15. NUMBER OF PAGES

33

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std 239-18
298-102

CONTENTS

CROSS-PLATFORM DEVELOPMENT: A DIFFICULT NECESSITY

INTRODUCTION

Background

Overview of Cross-platform Development

Cross-platform development is the process of developing software that will run on more than one operating system. It has recently become important for many reasons. The reasons include trying to reach the largest audience possible with a piece of software, developing for a platform that is too expensive to purchase for all the developers, or an unknown or changing target platform for the software. The popularity of operating systems such as Linux (freely distributed, open source Unix) and FreeBSD has been increasing in recent years. Unix operating systems provide performance and reliability. WinNT provides a large customer base and a low cost solution when compared to other commercial Unixes. For these reasons it is desirable to support multiple platforms when developing new software.

Cross Platform Models

Currently three different models of cross-platform development are widely used: double source tree, single source tree emulation, and single source tree translation. Each of these approaches has pros and cons associated with it, which are briefly discussed here.

Double Source Tree

Double source tree development involves maintaining separate copies of the application code, one for each platform supported. The name is drawn from the fact that most applications that take this approach are written for MS Windows and Unix systems, therefore two source trees are maintained. The problem with this approach arises when a project is very large. Maintaining and debugging multiple copies of the same code can become costly and difficult. This model of cross-platform development is the least desirable when developing an application from the ground up.

Single Source Tree Emulation

Developing with the single source tree emulation model involves writing the application for one platform, then using some type of platform emulator to run the application on

Manuscript approved July 26, 2000.

other platforms. For example, writing a full Win32 application and using WINE (A free windows emulator) to run the application on Unix, or writing an Unix application and using Exceed, Hummingbird, or Nutcracker to run the application under Win32. This model leads to problems because many times the underlying libraries of a platform are closed source, so emulators do not always support all the functionality of the original platform. Also, since the application is relying on the emulator, a poor emulator could make the application appear unstable. Although the application may run fast on the original platform, the same would not be true for platforms running an emulator, where performance would be slow. Single source tree emulation is a better model than double source tree, but it still has problems and it is not recommended for large or complex applications.

Single Source Tree Translation

The single source tree translation model involves putting a layer of abstraction between the application and the underlying architecture. This layer allows the developer to write the application once, using a platform independent API. The abstraction layer then translates the method calls into calls that will work on the relevant platform. The advantage of this approach is that this layer is compiled directly into the application, which provides speed and stability. This abstraction layer is normally provided through the use of a cross-platform toolkit. The downside to this model is that the application is dependant upon the toolkit used, which makes it impossible to change the toolkit at a later time. Also, since the toolkit might not directly work with the relevant platform elements, the application could have a slightly different appearance than an application directly written for the platform. Compared to the other two models, single source tree translation is the best option for large applications that require high performance and stability on all platforms. This paper will focus on the single source tree translation approach.

Problem Areas

Writing cross-platform applications can be a difficult process depending on the approach that is taken. Many small issues that appear when working with multiple operating systems on multiple architectures complicate the task. Many of these issues will be discussed in this section.

Graphical Interface

Most modern software requires a Graphical User Interface (GUI). Unfortunately, finding a cross-platform GUI solution is one of the largest problems. It is a difficult problem because almost every operating system uses a different mechanism for displaying graphics and text to the user. For example, MS Windows provides the Microsoft Foundation Class (MFC) Library as an API to the display, whereas Linux (and most other Unix's) use the X11 windowing system. Traditionally, two sets of graphical code would be written, one for each OS. This approach would quickly become a problem when it comes to maintenance and when trying to create a common look and feel across multiple operating systems for a single application.

File Input and Output (I/O)

File I/O problems can be seen in simple things like directory separators. For example, MS Windows uses the '\' character, whereas Unix uses the '/' character. This problem can also be seen when looking at the drive-naming scheme. MS Windows uses letters (that is: C:\, D:\) but Unix 'mounts' everything starting from '/', no letters involved. This means all code that assembles paths to files would need to be duplicated for MS Windows and Unix platforms.

Advanced Data Types

The days of the 'char*' are long gone. Developers now want to use more advanced data types to manage and manipulate data. MS Windows provides the developer with the MFC library that has data types for such things as strings and lists, but since they are contained in the MFC, they are obviously not cross-platform. Unix platforms don't provide any such data types directly. A common set of data structures or libraries must be found that allows programmers to use advanced data types cross-platform.

Software Development Tools

Some of the most important items to software development are the tools to which the developer has access. Making a developer use different tools on different platforms decreases productivity because of the time required to learn the particular tool, as well as the time it takes for the developer to adjust to the switch. These tools include such things as compilers, editors, versioning systems, debuggers, make systems, and documentation generators. Without these tools a developer is helpless, but getting

these tools for all development platforms and getting them to work together can be a project in itself.

Context

Who Is the Special Projects Group?

The Special Projects Group (SPG) is a small team of programmers and researchers in the Effectiveness of Navy Electronic Warfare Systems (ENEWS) Program Office, Tactical Electronic Warfare Division of the Naval Research Laboratory. Our mission is to research programming techniques and develop E.W. mission planning software and analysis tools for the Department of Defense.

Why the Special Projects Group performed this Study

This project was undertaken for the following reasons:

1. SPG was beginning a new phase in development. Because of this, we wanted to make sure things were designed properly, from the ground up, which lead us to the realization that we needed to do research to determine the best underlying design for future applications.

2. At the same time SPG started researching, our customers (including the Department of Defense) began requiring that all software produced with their funding would have to run on the WinNT operating system. However, we wanted to continue supporting our original customers who are using high-end Unix machines.

3. Like any organization that relies on the products it produces to bring in funding, SPG had to target the largest audience possible. The number of machines that can run an application increases dramatically with support for the WinNT platform.

4. In the past SPG only supported the IRIX operating system. Most of our products were very graphically intensive and only SGI machines could provide the performance required. However, because of the rapid growth of personal computers and accelerated graphics boards, a need to support these types of computers suddenly arose. The most common operating system on these machines was MS Windows. Since most of our developers have become accustomed to writing applications on and for Unix, a sudden move to MS Windows

would be too difficult and require too much downtime while developers learned the environment. Therefore a solution needed to be found that would allow a gradual move of developers.

Techniques

Overview

To research cross-platform development, SPG took a very straightforward approach. Requirements were broken down according to categories, then for each category a list of products was assembled. For each product, we assembled a list of desired features. Then evaluated all the products according to their feature checklist to determine the reasonability of using the product for future development.

Categories Considered

Special Projects Group formulated the following list of categories that we consider important to software development. The list is in no particular order, but a cross-platform solution had to be found to fill each category.

Category	Description
Platform / Environment	The machines, file system, user setup, printer setup, and file locations.
Documentation Tools	Tools to generate and format documentation for cross platform use.
Toolkits	Kits that provide basic cross-platform development tools. They include everything from GUI components to collection classes and advanced data types.
Compilers	Compatible C++ compilers.
Editors	Text editors that provide more than basic functionality and are programming language aware.
Debuggers	C++ compatible debuggers.
Make Systems	Generate makefiles for different platforms, since all compiling is done on the command line.
Versioning Systems	Allow multiple developers to work on the same project at the same time.
Design Tools	UML compliant design tools.

Experience Details

Evaluation Procedure Description

Special Projects Group found that the best and most comprehensive way to evaluate the products that filled each category was to make a checklist for each category that contained all the features we felt were important. These commercial and freeware products were divided among our developers and each product was tested according to how it would be used during real software development, then each developer completed the feature checklist. The checklists were compiled into large tables and the best product was chosen.

If for some reason no tool could be found in a particular category, more extensive searching for products was conducted. If no viable tool could be found, SPG took on the task of developing that tool in-house.

Feature Checklists

The following section contains the feature checklists SPG assembled for each category. The lists contain the features we felt were important in the selection of the product. Also, all products were to be evaluated on their ability to run on WinNT, IRIX, and Linux, which are the platforms we felt that we will need to run our applications. Both IRIX and Linux pass the Platform / Environment checklist, and because of customer demand we were forced to support WinNT.

Platform / Environment Checklist

This category includes the machines, file system, user setup, printer setup, and file locations require for cross-platform development.

File Sharing	Share files between computers. Access common directories.
Network Aware	Be compatible with standard network protocols.
Secure	User levels and password limited access.
Print Sharing	Share a common printer, or print to a networked printer.
Remote Login	Allow remote login and administration.
Memory Protection	All applications execute in their own memory space; therefore renegade applications can't crash the operating system.
Stable	Long uptimes.
WWW Browsers	Has some graphical application to allow for reading of standard HTML.
PDF Viewers	Has some application to allow for reading of PDF files.
FTP Applications	Has some application that supports FTP. The application doesn't have to have a graphical interface.
Telnet Clients	Has some application that supports Telnet. The application doesn't have to have a graphical interface.

Documentation Tools Checklist

This category includes the tools to generate and format documentation for cross platform use.

Easy to Use	Any developer can generate documentation without having to use a manual for the utility.
Cross-platform Output	Generates output that can be read cross-platform. The preferred format is HTML or PDF.
Support All Platforms	Can be run on all platforms.
Search for Docs in .h and .cpp	Configurable to allow documentation to appear in either .cpp or .h files.
Clean Doc Style	Minimal effect on how the developer would normally comment code. No special macros or odd preprocessing required.
Support for Selected Toolkit	Can support the toolkit. For example, if the toolkit uses a lot of macros, the documentation tool can do macro expansion.
Recognize and support the entire C / C++ language.	Can handle namespaces, templates, classes, inner classes, etc.
Organizational Pages	Generate alphabetical and/or hierarchical pages for easy navigation.

Toolkits Checklist

This category includes kits that provide basic cross-platform development tools. They include everything from GUI components to collection classes and advanced data types.

OpenGL Compatible	Supports or at least allows OpenGL drawing in a widget (drawable component).
POSIX Threads Compatible	Supports or at least allows POSIX threads.
GUI Builder	Has an easy graphical GUI builder, but doesn't require one.
Documentation	Has cross-platform documentation, preferably in HTML.
Extendable w/ Custom GUI Components	Allow developers to extend the predefined widgets in some way to create custom widgets.
Delegate control to non-GUI Activities w/o Multithreading	Provides a method of doing computations that don't affect the GUI, nor prevent proper GUI updating.
C++ API	Pure C++ API. Designed in an object-oriented fashion from the ground up.
Single Library	All of the toolkit is compiled into a single library to make moving and linking the library easy.
High Performance	High speed drawing capabilities.
Support	Customer support, either via email or over the phone.
Proven and Tested	A project exists as an example of what the toolkit is capable of doing.
Limited use of Macros	Small number of macros. Doesn't make the code too hard to read or debug.
Easy to Learn	A good C++ developer can learn to use the basics of the toolkit in a day and can learn most, if not all, of the toolkit in a week. A simple tutorial application can teach someone all they need to know.
Fulfillment of Least Common Denominator of Widgets	At least provide all the widgets that are common across all platforms.

Compilers Checklist

This category includes C++ compatible compilers.

Support Namespaces	Support standard C++ namespacing.
Support Templates	Support standard C++ templating, both in classes and methods.
Support all Standard C++	ANSI compliant.
Support RTTI	Support Run Time Type Identification. A new C++ standard that makes identifying objects quick and simple.
Command line only Mode	The compiler doesn't depend on a GUI to be invoked and documentation exists which explains command line switches.
Generate Shared & Static Libraries	Ability to generate standard platform libraries, both shared and static.
Generate Native Executables	Ability to generate standard platform executables that behave according to the platform requirements.

Editors Checklist

This category includes text editors that provide more than basic functionality and are programming language aware.

Syntax Highlighting	Support C++ syntax highlighting.
Configurable Indenting	Indenting can be configured to meet group-defined standards.
Proper Integration into OS	Conforms to OS standards when running. (i.e., Win32 applications launched from the command line should return the command line).

Debuggers Checklist

This category includes C++ compatible debuggers.

GUI Interface	The debugger uses a graphical interface.
Step Capability	Ability to set breakpoints and execute one line of code at a time.
Memory Inspection	Ability to view the contents of any memory location.
Memory Violation Detection	Ability to catch a memory violation and report where it happened.
Performance Profiling	Ability to monitor execution time of individual functions, then report which functions are taking too long to execute.
Multi-thread Support	Ability to debug multi-threaded applications.

Make Systems Checklist

This category includes tools that generate makefiles for different platforms. These tools are needed since all compiling is done on the command line.

Support Multiple Compilers	Generate makefiles that are compatible with a various number of compilers.
Support Multiple Make Applications	Generate makefiles that are compatible with a various number of make applications.
Command line Only	Doesn't require a graphical interface to function properly.
Predefined Configuration Files	Developers only need to configure the make system once for an application.
No Make Knowledge Needed	Developer never directly has to write a make file or any component of the make file.
Support Toolkit	Provide support for the toolkit. For example, if the toolkit needs some type of preprocessing or anything of that nature the make system will generate appropriate makefiles.
No Underlying Compiler Knowledge Needed	Developer doesn't need to understand the compiler flags for the given OS.
Easily Changeable	New compilers and platforms can be added without having to recompile or reinstall the application.
Support Shared and Static Libraries	Generate makefiles that tells the compiler to make the proper library, shared or static.
Support Executables	Generate makefiles that tells the compiler to make an executable.
Support Debug Information	Generate a makefile that tells the compiler to include debug information when compiling.
Echo Current Command	Ability to echo the command it is executing

	so the developer knows what is happening.
Human Readable Error Messages	Any errors will cause a human readable string to be outputted which states where the problem occurred and what the problem is.
Query Developer for Information at Run Time	Support asking the developer questions when generating make files, so the files are generated according to the developer's specific system configuration.

Versioning Systems Checklist

This category includes tools that allow multiple developers to work on the same project at the same time.

Tag Support	Files can be tagged with a string so they can be retrieved in that state later.
Multiple Developers	Multiple developers can obtain and edit a copy of the code at the same time.
Easy to Use	Simple interface that allows a developer to learn the basic usage in a few minutes.
Network Aware	Store and retrieve files on local and remote machines, across multiple operating systems.

Design Tools Checklist

This category includes design tools that enable the creation standard UML diagrams.

Standard Unified Modeling Language Support, including Class Diagrams	Allows the developer to create and edit UML class diagrams that meet the UML standard.
Sequence Diagrams	Allows developer to create and edit Sequence Diagrams.
Use Case Diagrams	Allows developer to create and edit Use Case Diagrams.
Smart Layout/Drawing Engine	Lines and objects are drawn neatly and require little reorganization by the developer.
Reverse Code Engineering	Ability to read old code and create UML diagrams.
Printable Diagrams	Ability to generate a printable document.
Ease of Use	Easy to learn. A programmer should be able to feel comfortable with the tool after only one day of learning.
Stability	Feels solid and runs without problems.

Collected Data

More products were tested than are displayed here due to space limitations. The products that were close in comparison with each other are listed. Products that were quickly eliminated are not show.

Platform / Environment

	File Sharing	Network Aware	Secure	Print Sharing	Remote Login	Memory Protection	Stable	WWW Browsers	PDF Viewers	FTP Applications	Telnet Clients
Win95	X	X		X				X	X	X	X
Win98	X	X		X				X	X	X	X
WinNT	X	X	X	X		X	X	X	X	X	X
Linux	X	X	X	X	X	X	X	X	X	X	X
IRIX	X	X	X	X	X	X	X	X	X	X	X

Documentation Tools

	WinNT	IRIX	Linux	Easy to Use	Cross-platform Output	Search for Docs in .h and .cpp	Clean Doc Style	Support for Selected Toolkit	Recognize and Support Entire C / C++ Language	Support All Platforms	Organizational Pages
Kdoc		X	X	X	X	X	X	X	X		X
Doxygen	X	X	X	X	X	X	X	X	X	X	X

Toolkits

	WinNT	IRIX	Linux	OpenGL Compatible	POSIX Threads Compatible	GUI Builder	Documentation	Extendable w/ Custom	Delegate Control	C++ API	Single Library	High Performance	Support	Proven and Tested	Limited use of Macros	Easy to Learn	Least Common Denominator
Qt	X	X	X	X	X	X	X	X		X			X	X	X	X	X
wxWindows	X	X	X	X	X	X	X	X		X						X	X
GTK		X	X	X	X		X	X		X				X	X	X	
Amulet	X	X	X	X		X	X	X		X							
Zinc 5	X	X	X	X	X	X	X	X		X			X		X	X	X

Compilers

	WinNT	IRIX	Linux	Support Namespaces	Support Templates	Support all Standard C++	Support RTTI	Command Line Only Mode	Generate Shared & Static Libraries	Generate Native Executables
CC		X		X	X	X	X	X	X	X
VC++	X				X		X	X	X	X
Borland BCC	X			X	X	X	X	X	X	X
GCC		X	X	X	X	X	X	X	X	X
Intel	X			X			X	X	X	X

Editors

	WinNT	IRIX	Linux	Syntax Highlighting	Configurable Indenting	Integration into OS
XEmacs	X	X	X	X	X	X
WinEdit	X			X	X	X
NotePad	X					X
VC++ IDE	X			X		X
Borland IDE	X			X		X
Write	X					X
Jot		X	X			X
VI		X	X			X

Debuggers

	WinNT	IRIX	Linux	GUI	Step Capability	Memory Inspection	Memory Violation Detection	Performance Profiling	Multi-thread Support
Visual Studio	X			X	X	X			X
SoftICE	X			X	X	X			X
BoundsChecker	X			X			X		X
TrueTime	X			X				X	
Purify	X	X		X			X		X
Quantify	X			X				X	
PureCoverage	X			X					
Code Vision		X		X	X	X		X	
DBX		X			X	X			X

GDB	X	X	X		X	X			X
DDD	X	X	X	X	X	X			X
Borland	X			X	X	X			X

Make Systems

	WinNT	IRIX	Linux	Multiple Compilers	Multiple Makes	Command line	Predefined Config	No Make Knowledge	Support Toolkit	No Compiler	Easily Changable	Shared & Static Libs	Support Executables	Debug Info	Echo Command	Readable Error	Query Developer
GNU Configure		X	X	X	X	X	X	X	X	X		X	X	X	X		
Imake		X	X	X	X	X	X	X	X	X		X	X	X	X		
Tmake	X	X	X	X	X	X	X	X	X	X		X	X	X	X		

Versioning Systems

	WinNT	IRIX	Linux	Tag Support	Multiple Developers	Easy to Use	Network Aware
CVS	X	X	X	X	X	X	X
RCS		X	X	X	X		X

Design Tools

	WinNT	IRIX	Linux	Class Diagrams	Sequence Diagrams	Use Case Diagrams	Smart Layout / Drawing Engine	Reverse Code Engineering	Printable Diagrams	Ease of Use	Stability
GDPro	X			X	X	X		X	X		
Together	X	X	X	X	X	X	X	X	X	X	X
Argo	X	X	X	X			X		X	X	X
Rational Rose	X			X	X	X		X	X		

Results

Eliminating Choices

Most of the test went as expected, the product Special Projects Group thought would be the best was in fact the best. However there were a few surprises with some of the products that we thought would come out on top were quickly eliminated.

The Standard Template Library (STL) was quickly ruled out because it wasn't supported on all platforms. Also, MS Windows uses different header files for the STL, so all of the included header files would need to be duplicated when writing cross-platform applications.

VC++ and Intel compilers didn't fully support the C++ standard. Language features like templates and namespaces caused these two compilers to fail. Microsoft has recognized some of these bugs and a fix is promised, but there is no time line guarantee. The significance of these compiler problems warranted them being eliminated.

Limitations

After evaluating all the products, SPG realized that there isn't a cross-platform solution for every problem. This means that in some cases two separate products must be used, one for each platform. This is the case with compilers, debuggers and editors. No single application would work well cross-platform in any of those categories, however there are different products that can be found to do the same job on different operating systems.

We also found that cross-platform applications work on many operating systems, but can never be fully integrated into a single operating system. For example, it is not possible to write an application that uses Microsoft's COM/DCOM because those elements won't be supported on all platforms. Most cross-platform applications tend to half-integrate into many operating systems, but never fully integrate into one.

Final Choices

After evaluating all the products, the best one was chosen. Below is the list of products we felt best filled the category along with a description of why we chose that product.

Platform / Environment

Linux provided an inexpensive solution and when combined with Samba it is one of the most flexible and reliable operating systems we have ever used. It can be installed on machines ranging from high-end servers to old low-end personal computers. It has incredible uptimes and behaves like most standard UNIXes. This makes the transition easy for developers who have worked with UNIX before.

IRIX best supports the heavy graphics load that some of our applications require. It also offers great administration tools and support. It is solid, reliable and secure. It offers the best integration of tools to allow for productive developing.

WinNT provides a secure and stable environment to test Win32 applications. It can run most Win32 applications, which means there is a large amount of available software for the system. It also has the best networking support in the Win32 domain, which allows it to blend into an existing network and use devices like network printers and common file servers. WinNT is also the platform that most of our clients are using.

Documentation Tools

Doxygen was chosen because of the cross-platform support and clean documentation style. It was the only tool that worked well cross-platform and supported some more advanced features like generating inheritance diagrams and a CGI script for searching the generated documentation. Doxygen also uses standard JavaDoc type rules with extensions. It will create collaboration diagrams using a tool from AT&T Labs, as well as linking multiple libraries' documentation together to allow referencing external documentation from within a project.

KDoc was also a good tool, but it required the KDE libraries, which are not cross-platform. These libraries lead to problems on both IRIX and WinNT since they couldn't be compiled cleanly.

Toolkits

Qt was the toolkit of choice. It offered the largest collection of cross-platform graphical components as well as offering a large selection of data structures, including lists, stacks, etc. The support is great, with a one-day or less response time as well as a mailing list offering instant answers. Qt is open source, which allows for easy debugging

and the ability to fix a bug before it is acknowledge by Troll Tech. It provides not only data structures, but also elements like socket and file IO. It was proven to be extendable by the KDE project under Linux and compiles cleanly under many operating systems. The only downside to Qt is that the event delivery mechanism requires the use of a pre processor, which we discuss later.

wxWindows was also an impressive toolkit, but it was harder to learn than Qt and the support wasn't quite as good. If a free solution is a must, wxWindows is definitely the recommend toolkit.

Compilers

G++ is the compiler of choice under Linux because not only is it free, but it offers great support for the newer features of the C++ language, including namespaces, templates and RTTI.

Under IRIX, CC was found to be the best compiler. Since the manufacturer of the operating system develops the compiler, it is well integrated into the environment and has great support for templates. It also offers the cleanest and most detailed error messages when compiling goes wrong.

Borland's CBuilder was the compiler of choice since both Intel's and Microsoft's compiler failed the tests. Borland's compiler is incredibly fast and very accurate to the ANSI standard, including many things in the standard that all other compilers ignore.

Editors

Under UNIX, XEmacs was chosen because of its advanced features, including syntax highlighting, configurable indenting, configurable shortcut keys, etc. The editor allows multiple files to be opened at the same time and allows a server to be run to make launching the application quick. XEmacs integrates well into the environment and allows the developer to do things like compile and debug directly in the editor.

The selection of editors under WinNT was poor. XEmacs was our first hope, but the project to port XEmacs to windows is still in early beta and not nearly stable enough to be used full time. One of the few applications that proved useable was WinEdit. It allows

multiple documents to be opened, supports syntax highlighting, and like XEmacs, it allows the user to run a single copy of the application to make startup time quick.

Debuggers

Under Linux, DDD is the debugger of choice. The selection of debuggers is rather small and most are command line only. Command line compilers prove to be very difficult to learn and rather tedious to use. DDD offers a GUI for most command line debuggers, giving the developer the power of the command line in an easy to learn interface.

With IRIX, both DDD and CVD prove to be viable options. DDD offers all the qualities mentioned above, while CVD offers all of the above plus great integration with the operating system, since CVD is written by SGI. The interface to CVD is clean and easy to use and it is both reliable and stable.

WinNT offers a larger selection of debuggers, but only a few are very usable. The problem with most of the WinNT debuggers is that the interface is hard to use and very cluttered. The best application found was Borland's CBuilder. The advanced window drawing tools Borland uses allows a developer to drag windows in and out of the main window, making it easy to debug large applications since multiple windows can be opened, moved and minimized individually. The application is solid and very powerful.

Make Systems

SPG found no suitable robust make system that works on all the platforms. Each operating system provided an application to compute dependencies and invoke the compiler, but these applications needed to be feed very detailed information. This proved to be too tedious and difficult for our developers so we decided to take on the task of writing an application that would run cross-platform to generate makefiles based on simple configuration files. Our final product was called CrossMake and is freely available under the Freedom of Information Act.

Versioning Systems

Very few cross-platform, network aware versioning systems exists. The only one that really met our needs was CVS. Many graphical interfaces exist for CVS, but they are not required to use the application. The application runs on both UNIX as well as WinNT using the Cygwin tools. CVS is fully network aware which allows machines that can't

mount drives to still communicate with the CVS server. The application is free under the GNU Public License and fairly easy to use once the basic commands are understood.

Design Tools

Testing the design tools took a long period of time. The first tool tested was GDPro, which was a good tool, but after long periods of use we found it to be unstable and the layout algorithms used were poor, requiring the developer to constantly relay out the diagrams. The manufacturer of GDPro informed SPG that support for IRIX would soon be dropped because of a small demand.

Rational Rose, a very popular tool, was found to be too difficult to use and very unstable. The C++ code generated from the application was very disorganized with special tags strewn throughout to allow the application to reload the code later.

This led us to the choice of Together. Written in Java it works on all the platforms that java supports. It has the best layout engine we saw as well as a simple clean interface. It automatically generates code, the layout of which is fully customizable in a few plain text configuration files. The tool also supports generating documentation with GIF images of the design, which is a nice feature but we found it a little too slow for general use. Together will update existing class diagrams automatically by reading and parsing existing code. It supports macro expansion and can preprocess code before creating class diagrams to be sure all variables are macros. Overall it is a great application that is in very active development with new features being introduced regularly.

Liabilities

Special Projects Group tried to make the best choice in each category, however this led us to some liabilities that were unexpected and as far as we are concerned unavoidable.

Use of Java

One of the major problems that we kept running across was that a lot of cross-platform tools are written in Java. This introduces many problems because of things like slow performance, dependency on Sun to continue Java development, and making sure that all of the development machines have the same version of Java installed. In most cases

the pros of using the tool outweigh the cons of Java. This is how we justified most of our choices that rely on Java.

Use of Slots/Signals (Pre Compiler)

Special Projects Group decided to use the Qt toolkit, which uses a mechanism of slots and signals to manage event handling. Although the slot and signal design makes Qt easy to learn and understand, it also requires the use of a preprocessor to translate the slots and signals into standard C++ code. The pre compiler source is provided free from Troll Tech (producers of Qt), which means it would be possible to modify the compiler if for some reason we had to.

Borland and VC++ DLLs

One problem we found after testing is that most applications or developers that provide DLLs have compiled the DLLs with VC++. This is a problem when our applications are built with Borland's compiler. Borland does provide a utility to convert VC++ DLLs into a format that can be used in Borland made executables, but it isn't guaranteed to work and the developer must rely on this tool in order to build an application.

Conclusion

Summary

Cross-platform development isn't necessarily easy, but it can be made much easier by using the right tools. A large software application should be developed from the ground up with cross-platform development in mind since all of the components of the application will need to work on more than one final operating system. Getting all the tools a developer will need installed and configured could take days. This should be the first step in the implementation phase. This will help prevent the developer from having to stop and install a new piece of software or tool in the middle of the implementation phase.

Any time that a developer uses third party solutions, dependency upon the producer of that solution is inevitable, therefore when choosing a product the company producing the product should be evaluated just as much as the product itself. Things like the life expectancy of the company, quality of support, and cost of the product should be strongly considered.

Recommendations

One of the most important things to look out for is a compiler that doesn't fully implement the C++ standard. Things like namespaces and templates seem like simple things, but they can easily throw a compiler into a broken state. Be sure to test all your code on all the compilers that will be used as often as possible. Don't assume that something will work on one compiler just because it works on one or two others.

Compatibility is another large problem. If the application requires DLLs or shared libraries, be sure that the compiler will generate and use components created using the respective platform standard. This is shown in the "Borland using VC++ DLLs" problem.

There are many free, open source tools available now that out performs many of the commercial products. SPG strongly recommends that you consider and evaluate free software. When it comes to software, price isn't always proportional to quality.

Try to make the migration to new platforms as easy for developers as possible. This can be done by continuing development on older platforms even if they won't be supported

in the future, then slowly work the developers to the new platforms one at a time, so there is no down time during development.

Changes

If SPG were to do this test again, we would do a few things differently, including making the test application we wrote a bit more complex. We would test more of the advanced features of the compilers, toolkits, debuggers, and other tools. There were times when we tested something, claimed it worked fine, but under a different situation it would fail. Testing needs to be the highest priority to which most evaluation time should be dedicated.

Shared libraries made with different compilers was something that SPG never really though about until it was too late. If the evaluation were to be done again, a list of what compilers produce and can use standard DLLs and shared libraries would have been a valuable reference.

References

Web Resources

All web resources where current as of the writing of this document.

Environment

- SGI: http://www.sgi.com
- Samba: http://www.samba.org
- WinNT (Microsoft): http://www.microsoft.com/ntserver/
- Linux: http://www.linux.org
- Cygwin: http://sourceware.cygnus.com/cygwin/

Documentation Tools

- Kdoc: http://www.ph.unimelb.edu.au/~ssk/kde/kdoc/
- Doxygen: http://www.stack.nl/~dimitri/doxygen/

Toolkits

- Qt: http://www.troll.no
- GTK: http://www.gtk.org
- Amulet: http://www.cs.cmu.edu/~amulet/
- wxWindows: http://www.wxwindows.org
- Zinc 5: http://www.zinc.com

Compilers

- BCC (Borland/Inprise): http://www.borland.com/bcppbuilder/
- VC++ (Microsoft): http://msdn.microsoft.com/visualc/
- Intel: http://developer.intel.com/vtune/compilers/cpp/
- GCC: http://www.gnu.org/software/gcc/gcc.html
- CC (SGI): http://www.sgi.com/developers/devtools/languages/c++.html

Editors

- XEmacs: http://www.xemacs.org
- WinEdit: http://www.winedit.com

Debuggers

- DDD: http://www.gnu.org/software/ddd/ddd.html
- CVD (SGI): http://www.sgi.com/developers/devtools/languages/c++.html
- DBX: http://www.sgi.com/developers/
- GDB: http://www.gnu.org/software/gdb/gdb.html

Make Systems

- iMake: http://www.opengroup.org
- GNU Configure: http://www.gnu.org/software/autoconf/autoconf.html
- tMake: http://www.troll.no

Versioning Systems

- CVS: http://www.gnu.org/software/cvs/cvs.html

Design Tools

- Together: http://www.togethersoft.com
- GDPro: http://www.gdpro.com

Portability Guides

- C++ Portability Guide: http://www.mozilla.org/hacking/portable-cpp.html
- C++ Programming Style: http://www2.wildfire.com/~ag/Engineering/Development/C++Style/doc.html

SPG Developers and Researchers

- Michael Pilone
- Gregory Stern
- Brian Solan
- Thomas Diepenbrock
- James Durbin
- Daniel Pilone
- Brian Calves
- Lawrence Schuette